Rate this page: 24 users 4.0 out f 5

Best Practices for Creating Adaptive User Interfaces with the Mobile Internet To lkit

Microsoft Corporation

January 2002

Summary: This article discusses the best practices for creating mobile Web applications and device filters with the Microsoft Mobile Internet Toolkit that adapt to multiple mobile device displays for Personal Digital Assistants such as the Pocket PC, Web-enabled cell phones, pagers, etc. (20 printed pages)

Contents

Introduction

The Microsoft® Mobile Internet Toolkit (MIT) provides the tools for creating new applications that target mobile devices. This toolkit allows developers to create a mobile application that will be adapted to the display of multiple mobile devices: Personal Digital Assistants (PDAs) such as the Pocket PC, Web-enabled cell phones, pagers, etc. Developers receive additional help from Microsoft .NET, which simplifies the deployment process of mobile applications, whether the applications are intended for a single server or a large server farm.

These best practices will help you create more efficient device filters and mobile Web applications. The practices described in this paper are not all-inclusive, and should be used only with due consideration. By following best practices, you are well on your way to creating world-class mobile Web applications.

Mobile Application Architecture and Deployment Issues

A mobile application is one that a mobile device can access. Mobile devices vary greatly in their display size, display quality, available bandwidth, and supported protocols. The Mobile Internet Toolkit handles most of the burden of displaying the information on these devices for you. However, besides the display issue, there are other factors that make a mobile application different from a Web-based application.

Limitations of mobile devices and mobile networks include:

- **CPU cycles.** A device may have a very limited CPU. This slows the parsing of the information being sent to the device and the speed at which the device encodes and decodes encrypted communications.
- **Half duplex vs. full duplex radios.** Some devices cannot work in full duplex. This means that they either send or receive data, but not both at the same time. This increases the latency and affects error recovery.
- **Latency.** Most mobile devices implement power-saving protocols, and there are also regulations that set a limit to the Specific Absorption Rate (SAR). The SAR is a measure of the bi logical effect of a radio-frequency-

emitting device to ensure that cellular phones and ther mobile devices are safe. These limitations result in increased latency on those devices.

- **Thr ughput.** Although General Packet Radio Services (GPRS) mobile devices are available with theor tical maximum speeds of up to 171.2 kilobits per second (Kbps), most phones still use Circuit Switched Data (9.6 Kbps) technologies. This means that mobile devices are affected by protocol overhead and the amount of data being sent.
- **Unknown network connectivity.** A mobile device can go from a full-coverage zone to a one-way zone, or even an out-of-coverage zone, easily. The presence of tall buildings, bridges, and so on affects the quality of the coverage for the device.

You should consider the preceding limitations when you design and deploy a mobile application. The following are recommendations based on those limitations:

- **Optimize the responsiveness of your application as much as possible.** Long delays between queries might force the device to go into error recovery mode.
- **Make your application compatible with mobile device error recovery mechanisms.** After a time-out, a dropped connection, or for many other reasons, a mobile device attempts to fetch the page from the mobile application again. This is the equivalent of pressing a submit button twice with the same data in a normal Web application.

  Note The gateway usually handles this. Gateway-to-server connection is usually very fast.

- **Define the Time-to-Live value appropriately.** Various carriers often use proxy services and caching heavily to quicken response time. This can cause problems with your site if the Time-to-Live variable is not set properly for the needs of your application.
- **Emphasize the reliability of your application.** Most mobile devices are ill-suited to give feedback on the quality of your mobile application. This means that adding a "Contact Us" button or the typical link at the bottom of each page is not a strategy that you can use to help you find problems with your application. Use tools to find and fix dead links. Do some performance analysis to find and minimize bottlenecks. Monitor your application after it is released to the Web to ensure that performance is not degrading over time.
- **Override the default error messages to redirect the users to your call center or other customer care group in your company.** The redirect can be in the form of a telephone number or another application.
- **Your site should use Secure Sockets Layer (SSL) encryption to protect the security of your customers.** It is much easier to intercept mobile messages than it is to intercept land-based communications.
- **Optimize your site for certain devices.** The Mobile Internet Toolkit does a great job of displaying the right thing on the right device. However, as new devices are introduced or as performance tests indicate, it might be best in certain circumstances to adjust the kinds of data that a device receives. For instance, a WML phone that supports images might have a wait time that is too long when the image is being sent, due to the Internet or the carrier's bandwidth. It might be best to remove those features so that users can enjoy a more responsive application. For more information about supported devices and performance issues, see "Requirements and Supported Devices" in the MIT online documentation.

Planning Phase: Design Goals

For any application, it is important to set the proper design goals during the planning phase. These design goals will dictate the best strategy to use for the deployment of your mobile application.

Much of the information provided in the Microsoft .NET Framework documentation is also relevant to you as a mobile application developer.

Designing a Mobile Application

You should keep in mind tw design goals when you distribute processing between the client and server. First, you want to minimize communication across the Hypertext Transfer Protocol (HTTP) connection. Regardless of h w fast

a connection is available, client processing will always be faster. You should pass inf rmation between the client and server only when abs lutely necessary.

A second design goal is to expose to the client only those server resources that are absolutely necessary to accomplish the processing task. Each request from the client should be fully qualified so that the server does not need to respond to the client for further information (and thereby increase round trips across the connection).

If a client can completely describe its capabilities to the server when it makes a request (for example, if the client can identify itself as a color-capable PDA), the server can immediately send a response that matches those capabilities rather than requesting further information from the client (for example, requesting whether the client supports color). This design concept can also be applied to the design of Web applications that support database access. For instance, if a client is checking the status of an order, you should provide the client with a **MyDatabase.Recordset** request that describes only that particular order, rather than all the records in the orders table.

Defining the Interface

When you create your application, be sure to take time to consider the target audience. When the application has both a desktop and a mobile presence, the desktop version should provide functionality to customize the mobile experience. Considerations include:

- Determine what makes sense for a mobile user. What provides the best value?
- Consider the fullness of the user experience:
  - Ensure that the user can easily enter and retrieve information.
  - Ensure that the application is customized for the target device and for the desktop.
  - Where possible, preselect the most common items and use them as default values. Default values increase user convenience, to reduce or eliminate the need for data entry based on key scenarios.
  - Consider the navigation and operational characteristics of the target device.
- Decide on your user interface design and begin by creating a storyboard for the application.

Building User Interface Components

The user interface can consist of menus, user input forms, and data display forms that may access collections of information. A good approach is to create a storyboard that details your application. Each storyboard becomes a form; each section becomes a page. Consider the following items for the UI:

- **Menus or a hierarchy of menus**. Provide the core navigation for your site:
  - **Static Menus**. Use the **List** control to display a static list of items. In many cases, a simple set of **Link** or **Command** controls is best.
  - **Dynamic Menus**. Programmatically generated choices provide a richer user experience, but may incur a higher overhead. For dynamic menus, a data-bound **List** control is most appropriate.
- **User Input**. On mobile devices (especially phones), use a step-by-step approach, as opposed to HTML pages that display everything at once. These desktop pages assume that the user has point-and-click capability to select and enter fields in any order. For mobile clients, a more directed approach is required. Some devices even force the user to step through what is on a user input form, in the exact order in which the controls have been laid out on the page. The original page may have a name field with a look-up button below it. The mobile application might need to place the name entry or look-up options at the beginning of the form sequence. For more information, see the Data Input Phase section later in this paper.
- **Data display forms (rendering)**. For large amounts of text, consider the use of pagination within forms.
- **User Input Forms**. You should not rely on pagination to break up multiple controls, but you can use it to paginate a single, paginating control like **TextView**, **ObjectList**, etc. In general, leave out pagination on forms and use small blocks of data. You can use the following guidelines:

- If a form ultimately has more than four r five controls, consider breaking up the job into multiple steps.
- If there's a step-by-step user input form, and some server processing or validation is required before the user proceeds to a subsequent step, this dictates a good place to break the form into two forms.

- **C llections**. The wise management of—and access to—data collections ensures a rich user experience. Collections are really another instance of showing a significant amount of data on a form. Hence, the same kinds of rules apply: Paginate the form and use a drill-down design to show shorter summaries first. You can use the following guidelines:
  - When you use **List** and **ObjectList** controls, if there are many items, consider using custom pagination, as opposed to automatic pagination, to allow for a better fit.
  - Support data tables by defining table fields that minimize complexity.
  - Reduce database access requirements and consider size limitations as a good practice for packaging and managing data within forms.
  - Use custom pagination to limit the number of items retrieved.

Data Access Alternatives

Another method of accessing data is to turn off view state and keep your own copy of the local data. Then, on each request, data bind to the local copy. For example, a user's inbox for mail might contain local copies.

Many data access designs have a cache layer available. As a developer, you may also add a user cache. This allows, for example, a one-time request for current news. Additional requests access the local cache.

Integration with Desktop Sites

Consider using redirection to send a user from one Web Forms page to another page that is matched to the user's browser capabilities. Suppose you have an existing desktop site, and you want to add a mobile site with the same URL. To do this, you could create a mobile .aspx file with the following code in the page load event:

```
if (Device.IsMobileDevice)
  RedirectToMobilePage("MyMobilePage.aspx");
else
  Response.Redirect("MyDesktopPage.aspx");
```

For examples of sharing code between desktop and mobile sites, see the IBuySpy Web site at www.IBuySpy.com.

Customized Views

By providing customized views of the application, you can create a better user experience that does not compromise the default rendering. As a general rule, when creating customizations, do not insert items that break the user interface model. Instead, do the following:

- Use **Header** templates to add **Home** buttons or other navigational elements. You can use templates to enhance the appearance and customize controls for specific types of hardware.
- Use **Content** templates to insert static descriptions (such as dividers) into the module.
- Use the **Details** template within **ObjectList** controls.
- Both menus and collections use **List** and **ObjectList** controls.
- Use templates to add rich customization. For example, use the **Alternate Item** templates for display (such as images of different resolution and size).
- Use common header and footer templates to provide a common appearance for your forms. Use style sheets within templates to define default or alternate characteristics for each custom device.
- Use property verrides to cust mize "d wn"—for example, to create custom labels and abbreviations for devices with limited display space.

Perf rmance

When designing adaptive user interfaces, developers consider the same metrics that apply to general Web applications:

- Requests per second
- Time To First Byte (TTFB) and Time to Last Byte (TTLB)
- Processor use
- Scalability with respect to number of clients and processors
- Size response bytes

For more information, refer to Collecting Performance Data in Duwamish Online.

General Procedures for Improving Performance

The following suggestions may help improve the performance of your application:

- Remove the need for extra user actions, such as a Welcome page banner. Keep the user experience easy and efficient.
- Remove WML headers. The header forces an extra line during rendering.
- When using **ObjectList** controls:
  - Use multiple table fields. For example, place **name/qty** on one line.
  - Turn off wrapping. Use the short labels for text completion hints.
  - Set `BreakAfter ="false"`. Allow the **name** field and associated text box to appear on one line. You will need to revise the HTML source to add needed spaces.
- Use a **Panel** control as a placeholder for dynamically generating controls. If there are no WML headers, add a link after inserting the **Panel** control to ensure separation.

Testability

When you design adaptive user interfaces for mobile devices, always include a "lowest common denominator" browsing capability so that you can preview your application design.

There is no substitute for using the actual devices that you are targeting for testing. For a list of devices, see "Requirements and Supported Devices" in the online documentation.

If possible, publish the application on the Internet to provide real-world access to the user base.

Globalization and Localization

A globalized application supports localized user interfaces and regional data for all users.

You may want to adjust the default encoding for specific devices and language configurations. The primary way to set the encoding type is by adjusting the **<globalization>** values in machine.config. The two primary settings available for adjusting the default encoding are:

- **requestEncoding**. The assumed encoding of the request (query string and form data).
- **responseEncoding**. The encoding for the content sent to the client.

The following example demonstrates how to set these attributes in machine.config:

```
<globalization
  requestEncoding="utf-8"
  responseEncoding="utf-8"
/>
```

Note that this change affects the entire computer. If you want to change only a single Web application, place a

modified <globalization> section into the application configuration file (web.config) file at the virtual root for the application.

For more detailed information about developing world-ready applications, refer to Building .NET Framework Applications in the Microsoft .NET Framework Developer's Guide.

Rendering Phase

Detailed discussion of rendering within the Mobile Internet Toolkit is discussed in the online documentation and may also be addressed in a future paper. This section introduces output caching practices and device-specific rendering.

Output Caching

Microsoft ASP.NET Web Forms pages include automatic support for caching page output. A page can be cached by means of an **@OutputCache** directive.

In mobile Web Forms pages, the target device must vary cached output. For example, if a device running Microsoft Internet Explorer for the Pocket PC requests a page, the resulting output should be cached and returned only for other devices running Internet Explorer for the Pocket PC.

By default, the HTTP user agent string varies cached mobile Web Forms pages. However, other devices may have their output varied by additional properties. For example, a device with a single user agent string may have multiple screen-size settings, each of which may have different output. To allow for these variations, the page adapter can override the **CacheVaryByHeaders** property.

ASP.NET Web Forms user controls also support an **OutputCache** directive that allows their output to be individually cached. This is known as partial caching. However, user controls in mobile Web Forms pages do not contain this directive. Mobile Web Forms pages do not support partial caching, because the output of a user control can be different depending on the contents of the rest of the page.

Device-Specific Rendering

Although Mobile Web Forms pages are capable of rendering to a variety of devices automatically, they also provide multiple means of specifying content specific to a device or class of devices, allowing the developer to customize a page to take advantage of particular features of a device.

Device Filter Considerations

The following notes apply to the creation and management of device filters:

- The Mobile Internet Toolkit distinguishes device filters by using both the name and argument value of each filter. As a result, two device filters can have the same name if they have different argument values.

  Note This applies to device filtering as exposed through the Mobile Internet Designer.

- The applied device filter named Default always results in a successful evaluation. If present, this filter blocks all other evaluations below it in the list. This filter will, therefore, logically appear as the last device filter in the Applied Device Filters list. However, applications can use the Default filter to catch every device not matching any of the filters that precede it in the list—in other words, to define an applied device filter that does not refer to any device filter saved in the application configuration file.
- Special care should be taken when renaming device filters that are already applied to a control. Renaming applied device filters does not rename the definition in the web.config file or in the code-behind file, and you will lose the logical association between them.
- Device filter names are case-sensitive.

Data Input Phase

When you design for data input, consider ways to enable the user to efficiently and intelligently enter and select data.

Regarding user input on mobile devices (especially phones), use a step-by-step approach, as opposed to HTML pages that display everything at once. These desktop pages assume that the user has point-and-click capability to select and enter fields in any order. For mobile clients, a more directed approach is required. The original page may have a name field with a look-up button below it. The mobile application might need to place the name entry or look-up options at the beginning of the form sequence.

Also, consider scaling issues. The size of the data per page may affect the number of server hits.

To elicit user input, use the **SelectionList** control in the following ways:

- Provide default values, such as the current date, whenever possible. Pre-fill text, such as current news and quotes.
- Use multi-step forms to determine the rest of the forms.
- Because scripting is not generally supported, you may need to insert into the form a **Next** command that results in a server-side action to activate the next form.
- Consider intermediate steps. Keep each form as simple as possible.
- Consider using the show/hide capabilities to show and hide different controls on a form (to achieve the affect of multiple forms).
- WML is not supported due to issues of backward compatibility.
- Some forms require server logic (to determine regional data and availability).
- The server hit is almost always bigger than the cost of processing on the server. After that, any back-end database access will far exceed any rendering costs. Rendering is the next largest processing overhead.
- Use the **ObjectList** control to view information from a database in tabular format.

Navigating Mobile Applications

This section describes the limitations and special considerations for navigating within an application on mobile devices, and best practices when converting existing applications.

For example, when you adapt an existing HTML application, there is a temptation to look at the toolbar first and convert each toolbar into a link on a form. A better way would be to create a **List** control with each toolbar item represented as a link in the list.

When paging through a paginated list, you can use the **Form.CurrentPage** property to mark a specific page and to go to the page.

Being Security Conscious

The .NET Framework provides a code access security model that allows administrators to modify security policy to meet individual needs. Improperly administered code access security policy can potentially create security weaknesses. For more details, refer to Code Access Security Model in the Microsoft .NET Framework Developer's Guide.

Mobile Web Forms controls depend on the .NET Framework and can be rendered unusable by changes in the .NET Framework. Versions of the mobile controls and the .NET Framework must be matched to ensure interoperability.

Protecting Resources

Make sure that "cookieless" sessions are turned on. Cookieless state management is required to address those devices that do not support cookies. On Web farms, use SQL session state, unless you are not using view state and session state.

It is important to use `MobileFormsAuthentication.RedirectFromLoginPage` instead of `FormsAuthentication.RedirectFromLoginPage`, and to use `MobileFormsAuthentication.SignOut` instead of `FormsAuthentication.SignOut`. If you use the wrong signature, cookieless and UP.Link-enabled devices will not be signed out.

MobileFormsAuthentication works in conjunction with **FormsAuthentication** from ASP.NET. To authenticate a user, you can authenticate against the **<credentials>** section in web.config, or you can provide custom authentication.

For example, you can authenticate a user against user names and passwords in a database, or you can authenticate a user against the Microsoft Active Directory™ directory service. After the user is authenticated, you should call `FormsAuthentication.SetAuthCookie`, which places the authentication cookie into the `HttpResponse.Cookies` collection. You should then call `MobileFormsAuthentication.RedirectFromLoginPage`. If the device does not support a redirect with a cookie, **MobileFormsAuthentication** will remove the cookies collection. In both cases, the authentication cookie is appended to the URL as a query string variable. When the device makes another request and the authentication cookie is supplied, **MobileFormsAuthentication** stops appending the authentication cookie to the URL. If a cookie is not supplied, it is assumed that the device does not support cookies, and the authentication cookie will continue to be appended to the URL.

The following are sample files and code that can be used to perform **MobileFormsAuthentication**.

web.config

```
<authentication mode="Forms">
   <forms loginUrl="login.aspx" name="nameOfAuthCookie" timeout="60"
                  path="/" >
      <credentials passwordFormat="Clear">
         <user name="username" password="password"/>
      </credentials>
   </forms>
</authentication>
```

login.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
                  Language="c#" %>
<%@ Register TagPrefix="Mobile"
                  Namespace="System.Web.UI.MobileControls"
                  Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.Security" %>
<%@ Import Namespace="System.Web.Mobile" %>

<Mobile:Form id="formA" runat=server Paginate="True">
   <Mobile:Label runat="server">Enter username</Mobile:Label>
   <Mobile:TextBox id="txtUsername" runat="Server" Text="username" />
   <Mobile:Label runat="server">Enter password</Mobile:Label>
   <Mobile:TextBox id="txtPassword" runat="Server" Text="password" />
   <Mobile:Command runat="Server" OnClick="Login_Click"
                  SoftkeyLabel="Login" ID="Command1"
                  NAME="Command1">Go</Mobile:Command>
   <Mobile:Label runat="server" id="lblError" />
</Mobile:Form>

<script language="c#" runat=server>
void Login_Click(Object sender, EventArgs e)
{
   if(IsAuthenticated(txtUsername.Text, txtPassword.Text))
   {
      FormsAuthentication.SetAuthCookie(txtPassword.Text, false);
      MobileFormsAuthentication.RedirectFromLoginPage(txtPassword.Text,true);
   }
   else
   {
      lblError.Text = "Please check your credentials";
   }
}

bool IsAuthenticated(String user, String password)
{//Check the values against forms authentication store
```

```
    if(FormsAuthentication.Authenticate(user, password))
    {
       return true;
    }
    else
    {
       return false;
     }
}

</script>
```

Page1.aspx

```
<%@ Register TagPrefix="mobile"
                     Namespace="System.Web.UI.MobileControls"
                     Assembly="System.Web.Mobile" %>
<%@ Page language="c#"
                     Inherits="System.Web.UI.MobileControls.MobilePage"
                     AutoEventWireup="true" %>

<mobile:Form id=Form1 runat="server">
   <mobile:Label id=label1 runat="server"
                     Text="Authenticated!"></mobile:Label>
</mobile:Form>
```

Add the three files to an Internet Information Server (IIS) application and browse to the Page1.aspx page. You will be redirected to Login.aspx and required to log on. After you successfully log in, you will be redirected to Page1.aspx.

Related Topics

Before you proceed with this white paper, it is recommended that you read the following documents:

- Microsoft .NET Passport SDK
- ASP.Net Web Application Security

Best Practices for Developing Mobile Controls

Best practices for developing mobile controls include the following:

- Combine multiple mobile Web forms on one page. Doing so makes it easier to share data between forms, minimizes the need to pass data from page to page, and makes more efficient use of the technology. If you attempt to place all forms on a single page, performance will suffer. Partition forms that are interrelated place each set of interrelated forms on a single page, separating out the unrelated groups onto individual pages. Keep in mind that every request requires the entire page to be read in again and all controls are re-constructed.
- Cookieless state management is required to address those devices that do not support cookies.
- To conserve resources, be selective in the choice of data type to ensure that the size of a variable is not excessively large.
- Keep the lifetime of variables as short as possible when the variables represent a finite resource for which there may be contention, such as a database connection.
- Keep the scope of variables as small as possible to avoid confusion and to ensure maintainability. Also, when you are maintaining legacy source code, the potential for inadvertently breaking other parts of the code can be minimized if variable scope is limited.
- Use only one transaction scheme, such as Microsoft Transaction Server (MTS) or Microsoft SQL Server™ and minimize the scope and duration of transactions.
- Be wary of using Active Server Pages (ASP) session variables in a Web farm environment. At a minimum, do not place objects in ASP session variables because sessi n state is stored n a single computer. Consider storing session state in a database instead.

- Stateless components are preferred when scalability or performance is important. Design the components to accept all the needed values as input parameters instead of relying on object properties when calling meth ds. Doing so eliminates the need to preserve object state between meth d calls. When it is necessary to maintain state, consider using alternative methods, such as maintaining state in a database.
- Do not open data connections by using a specific user's credentials. Connections that have been opened using such credentials cannot be pooled and reused, thus losing the benefits of connection pooling.

General Procedures and Best Practices When Using the Designer

The following notes apply as you develop your application by using the Mobile Internet Designer:

- Because the designer does not attempt to render pages as they would appear on any particular device, it does not use the **ItemsPerPage** property. Mobile Web Forms pages do not appear paginated in the designer. However, this property is used at run time.
- Styles are defined for your entire mobile Web Forms page, and only for that page. To define a style sheet for more than one mobile Web Forms page, you must set up an external style sheet.
- The template-editing mode in the Design view is supported only for HTML-based templates. You will need to use the HTML view to edit the contents of non-HTML templates. However, you can prepare a control for templating by using the **Templating Options** dialog box from within the Design view.
- The information specified in the **Markup Schema** field is used only by the designer and has no effect at run time. The schema will provide correct Microsoft IntelliSense® and syntax checking in the HTML view, as you type template contents.

General Practices

Other general design practices to consider:

- When possible, move the controls out of the template and have them apply to the entire form.
- Use property overrides within the templates to modify properties on a per-device basis.
- Use data binding over scripting whenever possible.

When controls are placed inside templates, any property (within a control) that is placed within a template can be data bound back to a structure in script. The structure can then be manipulated, as opposed to manipulating the control directly.

Data binding, however, does not address the issue of programmatic control of the page. The solution can be simplified and made more maintainable by using the **HasCapability** method, because the **<DeviceSpecific>** element already uses this method as part of its implementation.

F r WML-based browsers and devices that can handle multiple forms in the markup, you can link forms into a single deck instead of sending the user back to the server for each form request. When a form contains navigation elements that activate another form without interaction from the server, it is a linked form.

For example, Form B is linked to Form A if the following conditions are met:

- Form A contains a **Link** control with its destination as Form B.
- Form A does not contain an **OnDeactivate** event handler.
- Form B does not contain an **OnActivate** event handler.

If you have a link of the format `#form1` in a control contained in a user control, the **ResolveFormReference** method looks for a form with the value of the **id** property set to `form1` in the user control. If the form is not found, the **ResolveFormReference** method goes up the chain of any nested user controls, and then searches for the form on the page.

Syntax Notes

Scoping rules go up, not down, the hierarchy. If you want to link to a form that is contained in a user control, you

must use the following syntax to identify the form:

```
#mc1:form4
```

In the preceding syntax, mc1 is the user control identifier. The colon (:) separates the reference to the form.

> **Note** There is no support for element anchors (URLs of the form `page.aspx#element`, where *page* is not the current page).

Optimizing View State for Mobile Applications

For mobile Web Forms pages, the following considerations are important.

Saving view state to the session is highly optimized. If there is no view state to be saved, nothing is stored in the session, and no identifier is sent to the client. However, application developers who want to avoid using session management, or who want pages capable of high throughput, can consider reducing or eliminating the use of view state. In many application cases (such as rendering a page of formatted text), view state is unnecessary and is best turned off.

In addition to the application view state, there are other types of additional state information about the page that a mobile Web Forms page must store. This information might include the active form, or pagination information about the form. Such information is always sent to the client rather than kept on the server, and is usually generated in an optimized way. For example, if the first form is active, or the first page of a form is being shown, this information is not saved, because these are default states. Such state information is called the private view state. All controls can override the **LoadPrivateViewState** and **SavePrivateViewState** methods to read and write private view state.

Restrictions

As part of best practices, developers need to be aware of the constraints that are part of the Mobile Internet Toolkit. These restrictions assure the best outcomes.

Environment Restrictions

In the Mobile Internet Designer integrated development environment (IDE) for Microsoft Visual Studio® .NET, there are a few restrictions on how you can manipulate controls. Control violations can occur if, for example, you inadvertently drop a control in the wrong place or misspell a word in a **@Register** directive.

Control Restrictions

The designer does not prevent you from incorrectly dropping controls. However, it displays an error message when you do. Use the following guidelines to prevent errors when using the designer.

Always:

- Drop mobile controls on mobile Web Forms pages and not ASP.NET Web Forms pages.
- Place all controls, except the **Form** and **StyleSheet** controls, in containers or mobile user controls.
- Follow the cardinality and containment rules for controls.
- Ensure that the **@Register** directive is properly spelled if you have edited it.

Do not:

- Drop a mobile Web Forms control on a page that is not a mobile Web Forms page, such as a Web Forms page.
- Drop a Web Forms control or HTML server control anywhere on a mobile page other than in an HTML template.
- Drop a control of a particular type when the cardinality rules dictate that the number of controls of that type on a page or in a particular container have reached maximum. For example, multiple **StyleSheet** controls are not all wed on the same page.
- Delete **@Register** directives from m bile Web Forms pages.

Page Violati ns

A page violation occurs when a mobile Web Forms page that contains mobile Web Forms controls is loaded into the Visual Studio .NET IDE and the **@Register** directive is missing or incorrect.

A page that contains mobile controls is loaded into the Visual Studio .NET IDE with a correct **@Register** directive, but the underlying page is not of type **MobilePage**. In this case, the following error message occurs:

```
"This control only works in pages of type MobilePage."
```

Containment Violations

If an ASP.NET Web Forms control is dropped on a mobile Web Forms page outside a template, the following error message occurs:

"This page can only support non-mobile controls in templates. Please delete this control or move it into a template."

To correct the problem, delete or move the control.

The following actions also cause containment violations:

- A **Form** control is not placed at the top level of the page.
- A **Panel** control is placed outside a **Form** control or template.
- A **DeviceSpecific** control is placed outside a **Form** control or **Panel** control.
- A **StyleSheet** control is not placed at the top level of the page.
- Any other mobile control is placed outside a **Form** control, **Panel** control, or template.

Cardinality Violations

If more than one **StyleSheet** control is added to a page, the second **StyleSheet** control is disabled and renders with an error message. If the first **StyleSheet** control is deleted, the remaining **StyleSheet** control becomes the active one.

If more than one **DeviceSpecific** control is added to a container control, the second one is disabled and renders with an error message. If the first instance is deleted, the remaining control becomes the active one.

Debug Configuration Settings

This section describes the debugging-related sections of an application configuration file (web.config). For the most part, you will not need to change them. However, if you must customize the debugging environment for your project, use the following information.

Dynamic Debug Compilation

Set `<compilation debug="true"/>` to debug .aspx files:

```
<compilation defaultlanguage="c#" debug="true" />
```

Set this value to **false** to improve the run-time performance of this application. This can also be done as a per-page configuration through the **@Page** directive:

```
<%@ Page ..... Debug="True" %>
```

Custom Error Messages

You can specify a set of custom pages for errors. When an error occurs, ASP.NET determines whether the application is configured t show custom errors, and whether an appropriate error page exists. If an error page exists, ASP.NET redirects t the error page, passing a parameter that contains the riginal URL.

If you configure your application to show custom errors, but there is no custom error page set up, the Mobile Internet Controls Runtime sends the actual error to WML client devices. For custom error pages to work properly on WML devic s, add the following code to the system.web section of web.config:

```
<configuration>
   <system.web>
      <customErrors> defaultRedirect="genericerror.aspx"
                  mode="RemoteOnly">
         <error statusCode="500" redirect="InternalError.aspx"/>
      </customErrors>
   </system.web>
</configuration>
```

Set `useFullyQualifiedRedirectUrl="true"` for mobile devices that cannot handle the redirect response without a fully qualified URL specified in the response. (For example, i-mode devices require this to be set), as shown in the following code:

```
<configuration>
    <system.web>
        <httpRuntime
            useFullyQualifiedRedirectUrl="true"
        />
    </system.web>
</configuration>
```

Application-Level Trace Logging

Application-level tracing enables trace log output for every page within an application. Set `trace enabled="true"` to enable application trace logging. If `pageOutput="true"` is set, the trace information will be displayed at the bottom of each page. Note that when you use this debugging method, it is important to ensure that the default browser for Visual Studio .NET is set to the internal browser.

If you need to browse your application with an emulator while you are tracing, set `trace enabled = "true"`, as indicated previously:

```
<trace enabled="false" requestLimit="0" pageOutput="false" />
```

or

```
<trace enabled="true" requestLimit="40" pageOutput="true" />
traceMode="SortByTime" />
```

View the application trace log by using Internet Explorer to browse the file trace.axd from your Web application root directory.

> **Note** For Microsoft Visual C#® projects, **Unmanaged Debugging** is on by default; for Microsoft Visual Basic® projects, **Managed Only Debugging** is on by default.

> **Note** You may need to modify the **DEVPATH** environment variable to include the path where the link libraries reside.

Extensibility

Mobile Web Forms and mobile Web Forms controls offer the same extensibility features available in ASP.NET, and add support for working with multiple devices. Specifically, the following kinds of extensibility are provided:

- New mobile controls can be written and used in a mobile Web Forms page. New c ntrols can employ inheritance or composition to take advantage of existing controls.
- ASP.NET user controls can be used to write simple mobil controls declaratively.

- The output of any control can be customized on a device-specific basis by adding a new adapter for the control.
- Support for an entirely new device can be added by using adapter extensibility, without any changes to individual applications.

IBuySpy Portal and St re

The IBuySpy Portal Solution Starter Kit, found at www.IBuySpy.com/portal, demonstrates how you can use ASP.NET and the .NET Framework to build either an intranet or Internet portal application. IBuySpy Portal offers all the functionality of typical portal applications, including:

- Ten basic portal modules for common types of content.
- A "pluggable" framework that is simple to extend with custom portal modules.
- Online administration of portal layout, content, and security.
- Roles-based security for viewing content, editing content, and administering the portal.

All code contained in the IBuySpy Portal download package is free for use in your own applications. But if you prefer, you can customize the portal for your own use without writing a line of code. The portal includes built-in administration pages for setting up your portal, adding content, and setting security options.

Additionally, the IBuySpy Portal contains a number of modules that have been developed with the Mobile Internet Toolkit, providing portal functionality to mobile devices. In fact, visiting www.IBuySpy.com/portal with a Mobile Internet Toolkit–supported mobile device will present the mobile version of the portal.

Details of the IBuySpy Portal

The Default.aspx page performs a simple test for browser type and redirects either to a desktop or a mobile default page, as shown in the following code:

```
public void Page_Load(Object sender, EventArgs e)
{
     if (Request.Browser["IsMobileDevice"] == "true" ) {
         Response.Redirect("MobileDefault.aspx");
     }
     else {
         Response.Redirect("DesktopDefault.aspx");
     }
}
```

The default page is largely static—much of the work is done by other components. For example, the header, menu, and "popular items" list are all implemented in separate user controls referenced from this page.

The default page itself includes just a few lines of code to check whether the user has logged in. If so, the page displays a friendly, personalized welcome message. The personalization is done in the **Page_Load** event-handling method. This event is raised on the server every time a browser accesses the page. It provides a convenient way to structure server logic that needs to run when a page is accessed. The following code represents a default page:

```
        void Page_Load(Object sender, EventArgs e) {

            // Customize welcome message if personalization cookie is
                    present
            if (Request.Cookies["IBuySpy_FullName"] != null) {
                WelcomeMsg.Text = "Welcome " +
                     Request.Cookies["IBuySpy_FullName"].Value;
            }

        }
```

The welcome message is generated by retrieving a client-side cookie (persisted on the client in the Login.aspx and Register.aspx pages). If the cookie is found, the user name is extracted from it and displayed by using a server-side

label control. For details about how the cookie is generated, review the Login.aspx page.

Performance Notes

Because this page serves as the start page for the IBuySpy Store application—the page most heavily accessed—it has been specifically designed for top performance. We chose not t cache the entire page (unlike the **ProductDetails** and **ProductsList** pages) for output, because we want to be able to personalize the user's browsing experience.

For performance, the page is designed to avoid database calls for each request. Instead, we stored personalization information (in this case, the user name) in an optional client-side cookie.

In contrast, the **Menu** and **PopularItems** user controls do use information pulled from a database. However, these user controls use partial page output caching to cache the database results, and they update their data only once per hour.

Product Details

The results of this page are sent to the output cache to maximize performance. This is accomplished by adding an **@OutputCache** directive at the top of the page.

```
<%@ OutputCache Duration="60" VaryByParam="ProductID" %>
```

In this case, we elected to set the on output cache on the page with duration of one minute (60 seconds). ASP.NET will automatically store a separate cache entry for each different query string value—and will automatically render the appropriate output for subsequent requests.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, Visual Studio .NET, Mobile Explorer, Visual Basic, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contact Us | E-Mail this Page | MSDN Flash Newsletter

© 2002 Microsoft Corporation. All rights reserved. Terms of Use Privacy Statement Accessibility